Unit 3
C Programming

Lecture notes

Tokens & Syntax

- **The compiler collects the characters of a program into tokens.**
  - **Tokens make up the basic vocabulary of a computer language.**
- **The compiler then checks the tokens to see if they can be formed into legal strings according to the syntax (the grammar rules) of the language.**

Characters Used in C Programs

- **Lowercase letters**
  - **a b c . . . z**
- **Uppercase letters**
  - **A B C . . . Z**
- **Digits**
  - **0 1 2 3 4 5 6 7 8 9**
- **Other characters**
  - **+ - * / = ( ) { } [ ] < > ‘ “**
  - **! @ # $ % & _ ^ ~ \ . , ; : ?**
- **White space characters**
  - **blank, newline, tab, etc.**

The Six Kinds of Tokens in ANSI C

- **Keywords**
- **Identifiers**
- **Constants**
- **String Constants**
- **Operators**
- **Punctuators**

Keywords

- **Keywords are C tokens that have a strict meaning.**
  - **They are explicitly reserved and cannot be redefined.**
- **ANSII C has 32 key words.**
  - **Some implementations such as Borland’s C or Microsoft’s C have additional key words.**

ANSII C Keywords

auto	**do**	**goto**	**signed**	**unsigned**
break	**double**	**if**	**sizeof**	**void**
case	**else**	**int**	**static**	**volatile**
char	**enum**	**long**	**struct**	**while**
const	**extern**	**register**	**switch**	
continue	**float**	**return**	**typedef**	
default	**for**	**short**	**union**	

Identifiers

- An identifier is a token:
  - Composed of a sequence of letters, digits, and the underscore character _
    - Note: Variable names are identifiers
- Lower- and uppercase letters are treated as distinct.
- Identifiers should be chosen so that they contribute to the readability and documentation of the program.

Special Identifiers

- **main**
  - **C programs always begin execution at the function main.**
- **Identifiers that begin with an underscore should be used only by systems programmers**
  - **Because they can conflict with system names.**

The Length of Discriminated Identifiers

- **On older systems only the first eight characters of an identifier are discriminated.**
  - **identifier_one and identifier_two would be the same identifier.**
- **In ANSI C, at least the first 31 characters of an identifier are discriminated.**

Constants

- **Integer Constants**
  - **25 and 0**
- **Floating Constants**
  - **3.14159 and 0.1**
- **Character Constants**
  - **‘a’ and ‘B’ and ‘+’ and ‘;’ but not “a” or “B”**

Special Character Constants

- **The backslash is called the escape character.**
  - **The newline character ‘\n’ represents a single character called newline.**
  - **Think of \n as “escaping” the usual meaning of n.**
- **Enumeration constants will be discussed later in the course.**

String Constants

- **A sequence of characters enclosed in a pair of double quote marks, such as “abc” is a string constant, or a string literal.**
- **Character sequences that would have meaning if outside a string constant are just a sequence of characters when surrounded by double quotes.**
- **String constants are treated by the compiler as tokens and the compiler provides the space in memory to store them.**

Is it a String or Not a String?

- **“this is a string constant”**
- **“” /* the null string */**
- **“   “ /* a string of blanks */**
- **“ a = b + c; “ /* is not executed */**
- **“ /* this is not a comment */ “**
- **/* “ this is not a string “ */**
- **“ and**

  neither is this “
- **‘a’ /* a character, not a string */**

The Mathematical Operators

- **We looked at the mathematical operators briefly in the 3rd class:**

  + - * / %

- **In a C program we typically put white space around binary operators to improve readability.**

  a + b rather than a+b

The sizeof Operator

- **The C sizeof unary operator if used to find the number of bytes needed to store an object.**
  - **sizeof(object) returns an integer that represents the number of bytes needed to store the object in memory.**

printf()

printf(control string, other arguments);

- **The expressions in other_arguments are evaluated and converted according to the formats in the control string and are then placed in the output stream.**

```
printf("%-14sPayRate: $%-4.2f\n", "James Smith",
   8.95);
```

```
James Smith    Pay Rate: $8.95
```

- **Characters in the control string that are not part of a format are placed directly in the output stream.**

The Formats in the Control String

printf(“Get set: %d %s %f %c%c\n”,
1, “two”, 3.33, ‘G’, ‘O’);

- **%d Print 1 as a decimal number**
- **%s Print “two” as a string**
  - **“string” means a sequence of characters.**
- **%f Print 3.33 as a float**
  - **decimal or floating-point number**
- **%c Print ‘G’ & ‘0’ as characters.**

printf() Conversion Characters

Conversion character	How the corresponding argument is printed
c	as a character
d,i	as a decimal integer
u	as an unsigned decimal integer
o	as an unsigned octal integer
x,X	as an unsigned hexadecimal integer
e	as a floating-point number: 7.123000e+00
E	as a floating-point number: 7.123000E+00
g	in the shorter of the e-format or f-format
G	in the shorter of the E-format or f-format
s	as a string
p	the corresponding argument is a pointer to void; it prints as a hexadecimal number.
n	argument is a pointer to an integer into which the number of characters written so far is printed; the argument is not converted.
%	with the format %% a single % is written; there

Specifications

- **field width (optional)**
  - **An optional positive integer**
  - **If the converted argument has fewer characters than the specified width, it will be padded with spaces on the left or right depending on the left or right justification.**
  - **If the converted argument has more characters, the field width will be extended to whatever is required.**
- **precision (optional)**
  - **Specified by a period followed by a nonnegative integer.**
  - **Minimum number of digits to be printed for d, i, o, u, x, and X conversions.**
  - **Minimum number of digits to the right of the decimal point for e, E, and f conversions.**
  - **Maximum number of significant digits for**

printf ( ) Example

printf("Get set: %d %s %f %c%c\n",

1, "two", 3.33, 'G', 'O');

The first argument is the control string

"Get set: %d %s %f %c%c\n"

The formats in the control string are matched (in order of occurrence) with the other arguments.

Use of printf ( )

- **printf( ) is used for printing output. When printf( ) is called it is passed a list of arguments of the form:**

control string & other arguments

- **The arguments to printf( ) are separated by commas.**

Errors in printf ( ) Formats

- **A floating point format in a printf ( ) statement is of the form %m.nf**
  - **The value of m specifies the field width, not the number of digits to the left of the decimal point.**
  - **The value of n specifies the number of digits to the right of the decimal point.**
- **To specify two decimal digits to the left of the decimal point and three to the right, use %6.3f.**

Use of scanf()

- **scanf() is analogous to printf(), but is used for input rather than output.**
  - **scanf()in a program stops the execution of the program while you type something in from the keyboard.**

scanf ( ) Arguments

- **The first argument is a control string with formats similar to those used with printf().**
  - **The formats determine how characters in the input stream (what you are typing) will be interpreted so they can be properly stored in memory.**

Scanf **( )**'s Other Arguments

- **After the control string, the other arguments are addresses.**
- **Example: assume x is declared as an integer variable.**

scanf("%d", &x);

The & is the address operator. It says "store the value entered at the address of the memory location named x".

scanf ( ) Conversion

Conversion Character	How characters in the input stream are converted.
c	Character
d	decimal integer
f	floating-pint number (float)
lf	floating-point number (double)
Lf	floating-point number (long double)
s	string

A Peculiarity of scanf ( )

- **With printf() the %f format is used to print either a float or a double.**
- **With scanf() the format %f is used to read in a float, and %lf is used to read in a double.**

Another scanf() Peculiarity

- **When reading in numbers, scanf() will skip white space characters (blanks, newlines, and tabs).**
- **When reading characters, white space is not skipped.**

The Return Value of scanf()

- **When the scanf() function reads in data typed by a user, it returns the number of successful conversions.**
  - **scanf(“%d%d%d”, &first, &second, &third);**
    - **Should return a value 3 if the user correctly types three integers.**
    - **Suppose the user enters 2 integers followed by a string -- what happens?**
      - **What does our system do?**

Common Programming Errors

- **Failure to correctly terminate a comment.**
- **Leaving off a closing double quote character at the end of a string.**
- **Misspelling or not declaring a variable.**
- **Misspelling a function name.**
- **Omitting the ampersand (&) with scanf( ).**

Interrupting Program Execution

- **An executing program on a UNIX system can often be interrupted by entering a ^c from the keyboard.**
- **The kill command is another way of ending program execution.**
- **If your program is in an infinite loop you will have to use one of these methods to interrupt its execution.**

How the Compiler Handles Comments

/* This is a comment */

The compiler first replaces each comment with a single blank.

Thereafter, the compiler either disregards white space or uses it to separate tokens.

System Considerations

- **Syntax (Compile -Time) Errors**
  - **Syntax errors are caught by the compiler.**
  - **The compiler attempts to identify the error and display a helpful error message.**
- **Run-Time Errors**
  - **Errors that occur during program execution.**
  - **Memory errors caused by not using the address operator & with a scanf ( ) argument.**

Style

- **Use white space and comments to make your code easier to read and understand.**
  - **Indent logical subgroups of code by 3 spaces.**
- **Choose variable names that convey their use in the program.**
- **Place all #includes, #defines, main()s, and braces { } -- that begin and end the body of a function -- in column 1.**